

BEINGS: KNOWLEDGE AS INTERACTING EXPERTS

Douglas B. Lenat

Artificial Intelligence Laboratory
Stanford University, Stanford, California

Abstract

Knowledge may be organized as a community of interacting modules. Each module is granted a complex structure, to simulate a particular expert in some small domain. An extended analogy is drawn to a group of cooperating human specialists. Based on this, an internal constraint is imposed on the modules: Their structure must be standard over the entire community. Some advantages of a uniform formalism are thereby preserved. An experimental community was implemented for the task domain of automatic programming. It has managed to synthesize a few inductive inference LISP programs, nonformally, from specific restricted dialogues with a human user.

1. Experts and Beings

Consider an interdisciplinary enterprise, attempted by a community of human experts who are specialists in -- and only in -- their own fields. What modes of interactions will be productive? The dominant paradigm might well settle into *questioning and answering* each other. Instead of a chairman, suppose the group adopts rules for gaining the floor, what a speaker may do, and how to resolve disputes. When a topic is being considered, one or two experts might recognize it and speak up. In the course of their exposition they might need to call on other specialists. This might be by name, by specialty, or simply by posing a new sub-question and hoping someone could recognize his own relevance and volunteer a suggestion. Such transfers would be more common at the beginning, when the task is (by assumption) too general for any one member to comprehend. As the questions focus on more specific issues, single individuals should be able to supply complete solutions. If the task is to construct something, then the activities of the experts should not be strictly verbal. Often, one will recognize his relevance to the current situation and ask to *do* something: clarify or modify or (rarely) create.

What would it mean to *simulate* the above activity? Imagine several little programs, each one modelling a different expert. What should each program, called a *Being*, be capable of? It must possess a corpus of specific facts and strategies for its designated specialty. It must interact via questioning and answering other Beings. Each Being should be able to recognize when it is relevant. It must set up and alter structures, just as the human specialists do.

Let us return to our meeting of human experts. To be more concrete, suppose their task is to design and code a large computer program: a concept formation system[2]. Experts who will be useful include scientific programmers, non-programming psychologists, system hackers, and management personnel. What happens in the ensuing session? When an expert participates, he will either be aiding a colleague in some difficulty or else transferring a tiny, customized bit of his expertise (facts about his field) into a programmed function which can do something. The final code reflects the members' knowledge, in that sense. One way the session might proceed is for the specialists to actually *do* the concept formation task. As they become familiar with what part of their own expertise is being called upon, and in what ways, they can begin to isolate it. When it is clear

precisely what each is doing, they can take their extracted bits of knowledge, organize them, formalize them, and program them. {A conscious effort along these lines was made in [8], where experts gradually replaced themselves by programs. Instead of discussing how to write a speech program, they *did* speech recognition, until each one could introspect sufficiently into his own activities to formalize them.} For our task, one expects the psychologists to dominate the early discussions, later yielding to programmers. The project sponsor might be passive, submitting a single specification order for the program, or active, participating in the work as a (somewhat privileged) member of the team. This individual is the one who wants the final product, hence will be called the *user*.

How could Beings do this? There would be some little program containing information about CONCEPT-FORMATION (much more than would be used in writing any single concept formation program), another Being who knows how to manage a group to WRITE-PROGRAMS, and many lower-level specialists, for example INFO-OBTAINER, TEST, MODIFY-DATA-STRUCTURE, UNTIL-LOOP, VISUAL-PERCEPTION, AVOID-CONTRADICTION, PROPOSE-PLAUSIBLE-NAME. Like the human specialists, the Beings would contain far too much information, far too inefficiently represented, to be able to say "we ourselves constitute the desired program!" They would have to discuss, and perhaps carry out, the concept formation task. They would write specialized versions of themselves, programs which could do exactly what the Beings did to carry out the task, no more nor less (although they would hopefully take much less time, be more customized). This activity is referred to in the sequel as *automatic programming*. Some Beings (e.g., TEST) may have several distinct, streamlined fractions of themselves in the final program. Beings (e.g., PROPOSE-PLAUSIBLE-NAME) which only aided other Beings may not have any correlates in the final synthesized code.

An experimental system, PUP6, was designed and partially implemented. PUP6 synthesized a concept formation program (similar to [7]), but the user, who is human, must come up with certain specific answers to some of the Beings' critical queries. A grammatical inference program and a simple property list maintenance routine were also generated. A few new Beings had to be added to PUP6 to synthesize them.

The next section illustrates how the experts might have cooperated on the task of writing the concept formation program. Section 3 describes the program they produced. Next comes the Being hypothesis: complex but standard anatomy. Later sections explain this, both theoretically and by examining the behavior of the actual PUP6 pool of 100 Beings. The appendix exhibits a typical Being.

2. Experts Interacting

The input/output behavior of the desired concept formation program is specified in this section, and we eavesdrop on a simulated group of specialists as they get to work on writing it. As the presentation of the experts' activities becomes more specific, the reader's currently vague conception of Beings will be made less amorphous (because Beings are constrained to carry on approximately the same discussion as the experts below do).

Externally, the concept formation task can be specified as follows: pictures of structures (built out of simple geometrical shapes) will be presented one after another. For each such scene, the concept formation program, call it CF, must guess its name. The presenter will then reveal the correct name of the structure. CF must quickly learn to identify simple structures (ARCH, TOWER), and must never make the same mistake twice in a row. Assume, as given, a process which extracts a description of a visual scene.

Our group of experts are given this specification for CF. Assume that the user (the financial sponsor) is available for resolving important questions, via messenger, and he may in fact ask questions of the group. Whenever an expert speaks, almost all the others in the room hear him. Usually only a few can benefit from what he says, and fewer still care to react. The conversation in the room might go something like the following (the suggestive names of the experts are of course coincidental):

GENL-MANAGER: Can anybody here figure out what to do, what the user's saying? (waves the input/output specifications in the air)

PGM-MANAGER: I can. He wants a computer program to be written. If somebody will explain the task "concept-for-ma-tion" to me a little more clearly, I'll delegate it properly.

PSYCH: Permit me to assist you. I know all about concept formation. In fact, my master's thesis...

PGM-MANAGER: Wait, the user must be told that we'll be able to handle the job for him.

MESSENGER: Here, I can take that message. Go on with your work.

PGM-MANAGER: We need a name for this program. Somebody get one, please.

NAMER: How about "CONCEPT"? Maybe just "CF". Let's ask the user to decide.

MESSENGER: (panting) I just came back from there! Alright, I'm going... User says to call it "CF".

PGM-MANAGER: Now then, I have four people in mind, one of whom must take over now in an important way. Each of them always wants to do something different.

CHOOSE: Give me their names and I'll choose the best one for you.

PGM-MANAGER: They are INFO-OBTAINER, INFO-USER, PGMMER, and ANALYST.

CHOOSE: You four experts each must justify to me why you should take control now.

INFO-OBTAINER: We already have some unused information, so I guess I can wait.

INFO-USER: Let me have the floor! I know what to do with the user's specification.

PGMMER: Well, I suppose I could set up some "empty" top-level function, er...

ANALYST: Marginal profits accruing from my working at this point in time do not warrant...

CHOOSE: Okay, INFO-USER, you seem to be the only relevant expert.

INFO-USER: First thing is for PSYCH to tell us how to actually do concept formation.

PSYCH: There are several decisions to be made, depending on what your independent variables and your experimental setup are. For example, are we studying one individual, or must our program adapt to simulate many different subjects? Must stimulus items be just classified, or classified and ordered, or classified and ordered and metrized? Are the scenes left in view indefinitely, or just the current scene, or is even that one just flashed before the subject for a limited time? May the subject make written notes? Can he learn from positive instances of the concepts? from negative instances? Is there to be any positive transfer effect...?

DEFER: Hold on here; we can't swamp the user with lots of unnecessary questions. Let's go through each one and see

if it can wait. Consider the decision about the task being classificatory, comparative, or metrical. What is the first situation where that decision really matters?

WHEN-NEXT: This involves a list of alternatives. Who knows about that?

ALTERNATIVES: I know a few tricks here. If all the alternative sequences of activities have the same initial subsequence, then do that common subsequence before making the decision.

UTILIZE: In this case, all three begin by partitioning a domain of elements.

DEFER: Temporarily, replace the task CF by the task of partitioning a domain. (*The user is informed of this.*) Now consider the decision about subject-specific behavior being required.

PSYCH: This involves periodically inputting a description of the human subject.

CODER: That would mean adjusting the algorithms based on a vector of parameters.

WHEN-NEXT: This would probably affect the entire code.

DEFER: I can't defer this decision. Someone resolve it.

RESOLVE: Ask the user about it.

ASK-USER: Phrase this as a yes/no question. Explain each alternative to the user.

MESSENGER: That's my job... *<types on teletype>* User says no, don't simulate different people.

DEFER: The next decision...

Eleven decisions are ultimately proposed by PSYCH, and all but one are deferred.

INFO-USER: I have no objections now if someone wants the floor.

PGM-MANAGER: Do any of the other three experts I mentioned earlier want to speak now?

PGMMER: Yes. I think that the top-level function CF can now be coded.

CODER: Give me the arguments and the body of the code.

PGMMER: There are no known arguments. The body is a call on *PARTITION-DOMAIN*.

CODER: Okay. I will precede that with a call to an *INITIALIZE* function, and follow it with a call to a *FINALIZE* function, which are both defined as NIL for now. Is *PARTITION-DOMAIN* simple enough to be composed right now and filled in here?

MATHEMATICIAN: No way. Any realize...

CODER: Uh, thanks. There. The function CF is defined as: (LAMBDA () (INITIALIZE) (PARTITION-DOMAIN) (FINALIZE)).

ANALYST: Remind me to examine the initialization and finalization functions at the end of our task. If either function is still null, it will be deleted.

WARNER: I have just put that note into the code for CF, as a comment of type "demon"

PGMMER: Can someone advise me of what else to do to finish defining this function?

PGM-MANAGER: Each function should have a proper name. Show the user the names you have picked, and let him choose other ones if he prefers.

MESSENGER: Okay... The user agrees to all 3 names.

INFO-USER: Somebody, please tell the group how to do partitioning of a space of examples.

A complete script, like the above, was constructed by hand. In the sequel, this will be referred to as the *protocol*. In all, 87 different experts were called for: 17 specifically dealing with inductive inference tasks, and 70 dealing with programming, managing workers, and communicating with the user. Near the end of the protocol, the user is asked which of the three types of concept formation CF is supposed to do. He responds "*CLASSIFICATORY only*", and the experts discover that they are finished. All the newly created code is dumped out onto a fresh file. After hundreds of pages, a concept formation program meeting the user's specifications had been written. The next section will describe that program in detail.

3. The Program the Experts Wrote

One of the experts at the simulated meeting must have read P. Winston's dissertation[7], because CF, the synthesized concept formation program, was remarkably similar to the one therein described. CF has a much simpler graph-matching algorithm, and relations on relations are stored in a different way than simple relations on objects. Since CF was later synthesized by PUP6, the programmed pool of Beings, it is worth detailing here.

CF repeatedly scans a scene and tries to name it. As a first step, the scene is broken into a set of objects and a set of features (relations on those objects). CF maintains a model for each differently-named scene it has encountered. A model contains a description of the objects one expects in such a structure, a set of features which *must* be present in any scene having this name, a set of features which *must not* be present if the scene is to have this name, and a set of features which *may* be present or absent. Thus a model is an archetypal scene plus a name. For example, part of a scene might be described as:

OBJECTS a,b,c,d

RELATIONS (Green a) (Blue c) (Touches c d)
 (Supports a c) (Supports b c)

CF's current model for an arch might be:

NAME Arch

OBJECTS a,b,c

MUST (Supports a c) (Supports b c)

MUSTNOT (Touches a b)

MAY (Green a) (Wedge c) (Prism a) (Block b)
 (Parallel a b) (Red a) (Red b)

Each time it is confronted by a new scene, CF must scan its models until it finds one which matches it. A model is said to match a scene if all the MUST features associated with that model are observed in the scene, and all the MUSTNOT features are absent from the scene. CF informs the user of this guess, and accepts the proper name. If it guessed incorrectly, CF modifies its models. The wrong-guess model may have features added to its MUST or MUSTNOT sets. This is sufficient to prevent CF from making the same wrong guess twice in succession. The correct-name model may have to be modified or (if it's a new name) created and inserted into the list of models, to ensure that CF will eventually learn that concept. A *concept* here simply means a model; i.e., all scenes having a given name.

Suppose that the target program reads in the above scene fragment and tries to match it to the above ARCH model. The MUST relations should all be present. Yes, the scene does contain (SUPPORTS a c) and (SUPPORTS b c). Next, the MUSTNOT relations must be absent from the scene. Sure enough, (TOUCHES a b) isn't there. So the model and scene are consistent, and the program announces that its guess is ARCH. If the user verifies this guess, then the MAY set of the ARCH model is augmented with the relations (BLUE c) and (TOUCHES c d), and the OBJECTS set is augmented with "d."

If the user denies that the scene is an arch, CF sees if there are any relations in the ARCH model's MAY set which do not occur in the scene. If so, one of them (e.g., (PARALLEL a b)) will be transferred from the MAY to the MUST set. If no such feature existed, the program would look for a feature present in the scene but not mentioned in any set of the ARCH model (e.g., (TOUCHES c d)), and insert it into the MUSTNOT set. In either case, the user would be asked what the true name was, and that model would have its MAY set augmented by any new features in the scene and by any features on the true-name model's MUST or MUSTNOT sets which contradicted the scene.

4. Anatomy of Synergetic Cooperation

Consider the birth of one small idea necessary in the writing of CF (e.g., that of classifying a model's features into three categories (MUST, MUSTNOT, MAY)). No single specialist at the meeting could have had this idea by himself. How do intellects mesh, effectively communicate, and unite their powers? A tentative mechanism, which barely scratches the surface of this mystery, will be hypothesized. The Beings in PUP6 embody this concept, and are able to reproduce both the experts' discussion and the final CF program.

Viewing the group of experts as a single entity, what makes it productive? The members must be very different in abilities, in order to handle such a complex task, yet similar in basic cognitive structure (in the anatomy of their minds) to permit facile communications to flow. For example, each specialist knows how to direct a programmer to do some of the things he can do, but the specific facts each expert has under this category must be quite unique. Similarly, each member may have a set of strategies for recognizing his own relevance to a proposed question, but the *contents* of that knowledge varies from individual to individual. The hypothesis is that all the experts can be said to consist of categorized information, where the set of categories is fairly standard, and indicates the *types* of questions any expert can be expected to answer. An expert is considered *equivalent* to his answers to several standard questions. Each expert has the same mental "parts", it is only the values stored in these parts, their contents, which distinguish him as an individual.

Armed with this dubious view of intelligence, let us return to the design of Beings. Each Being shall have many parts, each possessing a name (a question it deals with) and a value (a procedure capable of answering that question). Henceforth, "*part*" will be used in this technical sense. When a Being asks a question, it is really just one part who is asking. In fact, it must be that the *value* subpart of some part can't answer *his* question without further assistance. He may not know enough to call on specific other Beings (so he lets anyone respond who feels relevant), but he should *always* specify what Being *part* the question should be answered by. By analogy with the experts, each Being will have the same "universal" set of types of parts (will answer the same kinds of queries), and this uniformity should permit painless intercommunication. Since the paradigm of the meeting is questioning and answering, the names of the parts should cover all the types of questions one expert wants to ask another. Each part of each Being will have implicit access to this list: it may ask only these types of questions. Each Being should *not* have access to the list of all Beings in the system: requests should be phrased in terms of what is wanted; rarely is the name of the answerer specified in advance. (By analogy: the human speaker is not aware of precisely who is in the room; when he feels inadequate, he asks for help and hopes someone responds). Another point is that Beings are not a recursive concept (like ACTORS[3] are): a part of a Being is a brief collection of knowledge (usually procedural), not another Being; a collection of Beings (also called a community, a pool, the system, or a group) is also not itself a Being. There are no *structured* clusters of Beings.

Once again: the concept of a pool of Beings is that many entities coexist, each having a complex structure, but that structure does not vary from Being to Being. This idea has analogues in many fields: transactional analysis in psychology, anatomy in medicine, modular design in architecture.

How can we test out this idea? We must build a pool of Beings, a modular program which will interact with a human user and generate the CF program. Recasting the idea into operational terms, we arrive at this procedure for writing a pool of Beings:

(1) Study the task which the pool is to do. See what kinds of questions are asked by simulated experts.

(2) Distill this into a core of simple questions, Q , in such a way that each inter-expert question or transfer of control can be rephrased in terms of Q . The size of Q is very important. If Q is too large, addition of new Beings will demand either great effort or great intelligence (an example of a system like this is ACTORS). If Q is too small, all the non-uniformity is simply pushed down into the values of one or two general catchall questions (all first-order logical languages do this).

(3) List all the Beings who will be present in the pool, and fill in their parts. The time to encode knowledge into many simple representation schemes is proportional to the square of (occasionally exponential in) the amount of interrelated knowledge (e.g., consider the frame problem). The filling in of a new Being is *independent* of the number of Beings already in the pool, because Beings can communicate via nondeterministic goal mechanisms, and not have to know the names of the Beings who will answer their queries. This filling in is *linear* in the number of Being parts listed in Q ; all parts of all Beings must be (at least, should be) filled in.

(4) The human user interacts with the completed Being community, until the desired task is complete.

Section 5 clarifies the effects of constraining that Q be constant (over all the Beings in the system). Theoretical aspects of Being systems follow, in section 6. Next comes an evaluation of PUP6's behavior. The uses and the problems with Beings are summed up in the final section.

5. Internal Details of Beings

A set of 29 ubiquitous questions were chosen, representing everything one expert might want to ask another. At least, they naturally encompass those questions which were asked during the simulated meeting, hence should be sufficient for generating C.F. Q , this universal set of Being parts, is listed in Appendix 1. The reader is urged to glance at this now, and refer to it whenever a Being part is specifically mentioned.

Each of the 100 Beings in PUP6 should have had a value for each part (in reality, only 40% of these 2900 slots were filled in; only 30% were actually necessary to generate C.F.). A value for a part is simply a LISP program which can answer that question, often by asking questions of the same Being, of other Beings, and of the user. A part may also assert some fact, create or modify some structure (including demons, Beings, and parts of Beings). Appendix 1 shows the values stored under each part for the typical Being named "INFO-OBTAINER".

The set of parts breaks into three rough categories: (1) those parts which are useful in deciding which Being gets control, (2) those which are used once the Being gains control, and (3) those useful only to answer the user's questions and keep him oriented. The next section describes categories 1 and 2; the section after that explicates the third category of Being parts.

5.1. Control in the PUP6 System

At the humans' meeting, only one expert spoke at a time; in the Beings community, only one Being has control at any given moment. He uses his parts to do things (ask, create, modify), and yields control either voluntarily or through interruption.

In slightly more procedural terms, the scenario is as

follows. One part of a Being senses its relevance (often the IDEN or EFFECTS parts, which are united with all such parts to form a large production system[5]). If more than one Being wants control at any time, a special Being, CHOOSE, seizes control momentarily. He asks each competing Being to evaluate its WHEN part, to see how seriously it needs to go immediately. If some Beings are still tied for first place, he asks them to evaluate their COMPLEXITY parts, to see which is the simplest. If any still tie for top, one is randomly chosen. In any case, the winner is then passed control. Once in control, a Being arranges some of its parts in some order and evaluates them. For example, the ARGS part might be first; if it asks for some arguments which no Being has supplied, then the whole Being might decide to fail. Some parts, when evaluated, might create a new Being, might ask questions which require this whole process to repeat recursively, etc. This "asking" really means broadcasting a request to one or two parts of every Being; for example "Is there a known fast way of gronking toves?" would be asked as a search for a Being whose COMPLEXITY part indicated speed, and whose EFFECTS part contained a production with a template matching "gronking toves". A list of the responders would be returned. (Incidentally, GERUND would recognize this, but later give up when no one could recognize "gronk toves".) The questioner might pose some new questions directly to these Beings, might turn control over to them directly, etc. One way or another, the Being eventually relinquishes control. If it had no direct successor in mind, all the Beings are asked if they want to take over. There will always be some Being who will take over; the general management types of Beings are always able -- but reluctant -- to do so.

How does each Being decide which parts to evaluate, and in which order, once it gains control? The answer might seem to be difficult or tedious for whoever writes Beings, since it might vary from Being to Being. In fact, it doesn't! The commitment to a universal set of Being parts is inefficient in some ways (each Being needed only a third of all the parts) but allows for some simplifications right here. What parts should be evaluated, and in what order, when a Being gains control? This decision depends primarily on the types of parts present in the Being, not on their values. But every Being has the same anatomy, so one single algorithm can assemble any Being's parts into an executable LISP function. Moreover, this assembly can be done when the system is first loaded (or when a new Being is first created), and need only be redone for a Being when the values of its parts change. Such changes are rare: experts are not often open-minded. The precise algorithm is sketched in the box below. The parts useful here include ARGS, DEMONS, META-CODE, COMMENTS, ARG-CHECK, and REQUISITES.

Assembling a Being into an executable function

When a Being B first gains control, its EXPLICIT-ARGS are bound. The IMPLICIT-ARGS are initialized, the name B is pushed onto the Being control stack, and any newly-activated DEMONS are so tagged. The Being who called B should have explained his reasons by assigning some phrase to the variable BECAUSE. This reason is now stored as a special sub-part of the WHY part of B . BECAUSE is rebound periodically in the META-CODE and COMMENTS parts, to keep current the explanation of each call that B makes. Each ARG-CHECK predicate is evaluated. If any returns NIL, the entire Being reports that it has failed; otherwise, the PRE-REQUISITES are examined. Effort is expended to make them true, if they are currently not satisfied. Each COMMENT is evaluated, then the CO-REQUISITES, META-CODE, and the current demons are executed in pseudo-parallel. Each POST-REQUISITE is then examined, and an effort made to satisfy it. The newly-activated demons are exorcized, B is

popped from the Being control stack, and the value computed by the META-CODE is returned. Some heuristics were devised to take advantage of the fact that the Beings often didn't need many of the standard parts. For example, INFO-OBTAINER has no new demons or requisites, so no parallel processing need be simulated.

5.2. Keeping the User Informed

In the earlier conversation excerpts, the simulated human user had no trouble whatever understanding what the simulated experts asked him. In the actual programmed PUP6 system, the human who was sitting at the teletype quite rarely understood what was wanted by the Beings. He frequently had to interrupt them and ask them questions about who was in control, why, what he was trying to do, what had recently transpired, etc. These ideally can be phrased as simple retrievals and EVALs of active Beings' parts. The Being parts most often called for by the user are the simple one-line "orientation" templates. These include WHAT, HOW, WHY, and AFFECTS. For theoretical reasons explained later, the synthesized program, CF, was written as a pool of Beings itself (by PUP6, but not during the protocol. Actually, a fortuitous "bug" in PUP6 created this intriguing situation.) Although its question-answering ability is inferior to PUP6, the fact that it has any such power was surprising to the author. In other words, one can interrupt the target program as it is running and ask questions. Any Being on the control stack will provide fully instantiated answers to any of its 29 allowable queries (its parts); all other Beings will provide only hypothetical answers. As an example, consider this actual excerpt of a human using the CF program synthesized by PUP6. (Some liberty has been taken with the English; e.g., the user really types *WHAT?*, not *What are you doing?*) "???" simply means "guess the name of the scene with these objects and relations". CF types in *italics*, the user in **boldface**.

CF: *Ready to accept brand new scene.*
 User: **??? (A B) (Block A) (Wedge B) (Touches A B)**
 CF: *Not an Arch. Not a Tower. Might be a Row.*
 My guess is Row. What is it Really?
 User: **Nothing**
 CF: *Transfer from the May to the Must set of the*
 Row model, the feature: (Block B)
 CF: *Ready to accept brand new scene.*

Here is the same excerpt, but with the user interrupting occasionally. "U:" signifies he has interrupted, and "OK." signifies "Go back to work now, CF."

U: **What are you doing?**
 CF: *Divide a domain into subdomains,*
 using specialized version of PARTITION-DOMAIN.
 How?
 CF: *By building up the partition gradually,*
 by adding one <scene, name> pair.
 OK.
 CF: *Ready to accept brand new scene.*
 User: **??? (A B) (Block A) (Wedge B) (Touches A B)**
 U: **What are you doing?**
 CF: *Break OBJECTS piece off input scene to examine.*
 What code is the current Being going to execute?
 CF: *The meta-code is (CADR arg1). Evals to (A B).*
 OK.
 CF: *Not an Arch.*
 U: **Why?**
 CF: *The ARCH MUST-relation (SUPPORTS A B)*
 is contradicted in the SCENE.
 OK.
 CF: *Not a Tower. Might be a Row. My guess is Row.*
 What is it really?
 U: **Why are you asking?**
 CF: *I must resolve it now, but can't, so I ask User.*
 OK. **What you wanted is: Nothing**
 U: **What are you doing now?**
 CF: *Go through space of concept names, and make user*

aware of changes.

What Beings might this affect?

CF: *MESSAGE is possibly called; some version of TEST*
 is possibly called.

OK.

CF: *Transfer from the May to the Must set of the*
 Row model, the feature: (Block B)

U: **Why?**

CF: *Change model of ROW until it contradicts*
 the previous inputted Scene.

OK.

CF: *Ready to accept brand new scene.*

6. Theory of Pure Beings Systems

We now discuss the constraints each Being, and each group of Beings, must conform to. Hopefully, ideas will be separated from implementation details, prejudices from plausible features.

It would be aesthetically pleasing to restrict all entities in the system to be Beings. However, this would cause an infinite regress, as each part of each Being would have parts which had parts... To stop this, one can assert that at some finite level, all constructs are primitive. ACTORS, for example, set this level to zero; Beings set it to one. ACTORS themselves are primitive, but only *parts* of Beings can be. For this reason, Beings can not be viewed as a convergent recursive definition, as can ACTORS.

Suppose it were decreed that the only autonomous entities possessing control abilities were Beings. In particular, we forbid any plain *functions* to exist. In the case of an automatic programming task, the Beings would have to write new Beings, not new LISP functions. The target program would thus itself be a community of Beings. In order to fill in all the parts, a vast amount of superfluous information would be collected. These supplementary facts can be viewed as a standardized, organized body of *documentation*, a formatted system of comments tacked onto each Being produced.

Which Beings would write the new Beings? Looking back at our interdisciplinary experts, we see that each expert is responsible for distilling his own essential contribution, which is then encoded by a programmer. Perhaps each Being should be able to direct construction of new, specialized Beings which relate to it. If no Being relates to a task, then it can't be coded; if several respond, they should cooperate. This ability is in reality the SPECIALIZATIONS part of each Being (see Appendix 1). The Being which actually does the creation (CODER) in the experimental system is almost trivial, getting very precise instructions from other Beings.

Since the pool must communicate with the user, some Beings must translate quasi-English phrases into calls on Beings. Drawing again on our experts analogy, we require that each Being recognize his own relevance. So translation is merely the act of asking the whole pool "Who can recognize this...", collecting the responders, having *them* decide who should take control, and letting the winner do the translation. Most communication is done as if it, too, were such a translation activity.

One bias is the rejection of debugging as a fundamental programming tool. It is felt to be worth the extra effort to make the system's internal model of the current partial target program *correct*. Debugging demands detective work, examining one's earlier efforts for flaws, for details which have been overlooked. Any tireless system should not ignore details, but rather defer them, asserting a warning to this effect when it does so. Procrastination is quite valuable; in PUP6, much effort is spent deferring any unresolvable decision. Undeferred unresolvable decisions must cause a backtrack point to be reluctantly set up. Another prejudice is that most carelessness bugs can be eliminated by this deferral, feed-forward, and precise record-keeping. Humans

depend on their adaptability to compensate for limitations in their brain hardware, but there is no need for an automatic programming system to do so. These biases are not inherent in the Beings formulation, but only in the design of the PUP6 system (and in the mind of the author).

To clarify what Beings are and are not, they are contrasted with some other ideas. FRAMES[4] are sufficiently amorphous to subsume Beings. In philosophy, FRAMES are meant to model perception, and intentionally rely on implicit default values; Beings intentionally avoid making decisions by default. This is also the difference between HACKER and PUP6. Since PUP6 writes structured programs, it should be distinguished from macro expansion. Macro procedures expand mechanically: $\text{expand}(\text{sequence } m_1 m_2) = (\text{sequence } \text{expand}(m_1) \text{ expand}(m_2))$. Beings could use information gleaned during expansion of m_1 to improve the way m_2 was handled. ACTORS[3], unlike Beings, have no fixed structure imposed, and do not broadcast their messages (they specify who gets each message, by name, to a bureaucracy).

Beings subsume (inefficiently) many popular AI features; the demonstration will be brief: A *demon* could be replaced by a Being whose ARG-CHECK predicate was the triggering predicate, whose WHEN part was high enough to ensure frequent attention, and whose META-CODE part was the body of the demon. An *assertion* in an associative data network could be a Being with only an IDEN part filled in; when it recognizes its relevance, a fully instantiated assertion is returned. A *function* is equivalent to a Being with only a META-CODE, ARGS, and EVAL-ARGS parts; one knows almost nothing about it before executing it. The inefficiencies should be clear: whenever a Being throws a question open to the floor, "Who can...", it takes an amount of time proportional to the number of Beings in the system. One would introduce this huge time factor by replacing any of the above mechanisms by Beings.

The number of Being parts seems to indicate the balance between uniformity and structure in the community. This was touched on as point (2) of Section 4. A small universal set of Being parts is necessary to preserve some of the advantages of uniformity (easy addition of knowledge to the system, easy inter-Being communication). This demands that the number of parts of each Being be, say, under 100. But it is the complex structure of a Being which makes complex behaviors feasible, including flexible communication as well as viable final products. So each Being should have many parts, say at least ten. This range, $10 \leq ||Q|| \leq 100$, is wide for the domain of automatic programming. In other domains, it may be narrow or disappear altogether; this would indicate that Beings could not be used effectively for those tasks.

7. Experimental Results

Two of the most significant questions for automatic programming systems are what programs are synthesized, and what kinds of things the user must tell the system.

7.1. Character of the Dialogue between User and PUP6

By lumping all the parts of the PUP6 system together conceptually, the interaction is seen as a *dialogue* between a human user and an automatic programming system. The level of detail maintained in the earlier polylogue (discussions among the experts) obscured what the human actually saw and responded. Below is the same excerpt as was presented in Section 2, but as the user sitting at a teletype actually saw it. These were the initial few lines of an actual 300-page long dialogue. In section 2, this excerpt was presented the way experts (the same way the PUP6 Beings) discussed it

PUP: I want any task.

USER: Write a program which does concept formation

PUP: I am about to write a program for the task:
specialized kind of CONCEPT-FORMATION.

PUP: Give me a proper name for this program.
(I suggest CF or CONCEPT)

USER: CF

PUP: Must CF mimic different human subjects?

USER: NO

PUP: I am temporarily replacing the task "specialized kind of CONCEPT-FORMATION" by the simpler task "specialized kind of PARTITION-A-DOMAIN".

From the above, one gets the true impression that the dialogue was fairly tedious: choosing names and resolving design decisions was the user's main role. There were, in sum, about 300 such responses involved in generating CF. 13 crucial responses were also necessary, where the user supplies needed information that was not explicitly requested by the Beings. A few of these key lines are: *Write a program which does concept formation; An "element of the domain of stimuli" is a static scene; The guessing process is one of searching, not computing; Terminate the loop iff any model feature is incompatible with the scene's features.*

The translation mechanism is simply the IDEN parts of all the Beings. Each such part contains a few productions, which are united with those of the other Beings into one large production system. The left side of a production is a template which is matched against the given phrase. The right side is a small program to be executed in case the phrase is recognized. Generally, that program simply says to translate some subphrase of the phrase, and then (probably) pass control to the Being which this is in the IDEN part of. While not up to the "state of the art" in natural language processing, this is both adequate and faithful to the spirit of distributing problems over all the Beings, by making them *parts*, so it is each Being's duty to know a little about them.

During the course of the dialogue, PUP6 originates the idea of a MUST/MUSTNOT/MAY structuring of each model's features. Here is how the user sees that portion of the synthesis. Notice that only the first user comment is other than a simple choice.

PUP: How do we know when to terminate this loop?

USER: Terminate iff any feature in model-features is incompatible with scene-features

PUP: PUP wants USER to type in name for specialized version of CONTRADICTION

USER: Call it IMPOSS

PUP: Move cursor to IMPOSS type of CONTRADICTION. I can no longer defer this decision: A: Contradiction arises from features who have probability=1 of occurring. B: Contradiction arises from features who have probability=0 of occurring. C: Contradiction arises from features who have probability>0&<1 of occurring Please type some of: A,B,C; or type: Defer, ?, or Details.

USER: A, B, and C are all relevant.

PUP: I recommend that each model's features be structured (along 1 dimension) into these 3 pieces: PRESENT, which will hold features whose occurrence should have probability=1; ABSENT, which will hold features whose occurrence should have probability=0; and EITHER, which will hold features whose occurrence should have any other probability. Please type back YES, NO, UNSURE, DETAILS, and/or 3 new names.

USER: YES. Call them MUST-BE-PRESENT, MUSTNOT-BE-PRESENT, MAY-BE-PRESENT.

PUP: Too long Nicknamed Must, Mustnot, May.

PUP: Located 3 references to model's features in code generated so far. In each case, the reference has been replaced by "(APPEND Must Mustnot May)".

This is PUP6 at its best: making just the "right" inference at the "right" time. The hundreds of inter-Being transfers are masked: the system appears to type out just what is necessary. The next few paragraphs dispell this false illusion; there is no mystery to PUP6's doing the right things, and any *typical* dialogue excerpt would be boringly verbose.

Because of its genesis from a single "experts meeting" protocol, the PUP6 pool of Beings was (i) easily able to reproduce that "proper" dialogue, but (ii) incapable of widely varied dialogues with the user. Two assumptions implicit in using the simulated experts' discussion as a literal model turned out to be serious: the abilities of any actual user of PUP6 must coincide with those of the user who was simulated in the protocol, and the order of and wording of the actual user's comments must closely adhere to those of the user in the protocol. Such difficulties must be overcome in any system designed for wide usership, but were considered ignorable in the specific automatic code generation task at hand.

Also as a result of this approach to system specification, each Being had only those parts specified which it actually would need in the ensuing dialogue. Part of the difficulty with new dialogues stemmed from this minimal completion. In the protocol, when a decision was made by experts, the knowledge necessary to follow the *other* alternative branch was not used, nor were such superfluous facts supplied to the Beings in PUP6. Thus the user of PUP6 must almost always resolve each choice the way the simulated (protocol) user did. It is felt that if all the parts of all the Beings had been faithfully filled in, this problem would have subsided. Basically, the difficulty is one of modelling all the possibly relevant knowledge an expert has, rather than (as was done) just capturing enough of his knowledge to do a few given tasks.

While all the Beings' interactions were invisible to the user, the system still swamped him with data about what was going on. For example, most of the entities he was asked to name were never referred to again by name. The converse problem existed as well: it was necessary to include a Being which simulated forgetfulness, to prevent, e.g., anaphora spanning minutes of real time. Orienting the user was not solved satisfactorily. Pointers into a graph of generated code were simulated, but often a user wished to refer to a piece of code not by name or by pointing, but by some brief meaningful (to him only!) phrase.

7.2. The Range of Programs Synthesized by PUP6

The system, PUP6, did eventually synthesize CF, the target concept formation program. PUP6 was 200 pages of INTERLISP[6]. CF was 30 pages long (6 pages when coded by hand during the protocol). CF was generated in 60 cpu minutes (compiled, PDP-10 TENEX). The dialogue consisted of 300K characters typed by PUP6, and 4K by the user. It occupied 300 pages, and five hours of real time.

Despite the lack of dialogue flexibility, it was felt that most of the Beings could be useful in generating other programs. For this reason, two additional target programs were specified. They were synthesized with little change to PUP6, but only by someone familiar with the system.

The second target program, G1, is a grammatical inference program, which accepts strings labelled LEGAL, ILLEGAL, or ??. In the latter case, G1 must guess the legality. Internally, potential sets of rules are maintained. Of the original pool, 49 out of the 87 Beings were used in synthesizing both targets. Four totally new Beings had to be added, related to formal grammars and rules. Unfortunately, the addition of any new Beings demands that the user be

acquainted with the format conventions of PUP6. The G1 program generated was 20 pages long; a hand-coded version was one-fifth that size.

PL was the final target program attempted, a simple property list manipulator. It repeatedly accepts requests from the user to insert, inspect, or delete some record(s). Any unspecified fields are treated as don't-cares, so a simple pattern-matcher had to be synthesized. Two Beings had to be added to PUP6. The important piece of data is that about half of the original PUP6 pool of Beings were actually used in *all three* target-synthesizing dialogues.

As proposed in Section 6, the Beings generate other Beings, never plain functions. This explains the huge increases in target code lengths in the PUP6 versions compared to the versions produced by hand when simulating the experts (who wrote the target programs as functions). CF was a pool of 56 brand new Beings, G1 37, and PL 24. As with PUP6, one can interrupt the target programs as they are running and ask questions. Any Being on the control stack will provide fully instantiated answers to any of its 29 allowable queries (its parts); all other Beings will provide only hypothetical answers. Recall the excerpt from CF itself running, found in section 5.2.

Some of the difficulties stem from the nature of the task. In any long dialogue, the user often forgets, changes his mind, errs, etc. A very sophisticated user model would be necessary to accommodate this errorful process in a non-debugging system. Without such abilities, the system itself may be led into error. While most bugs are avoidable by careful record-keeping, it proved unrealistic to make no provision for debugging a new thirty-page program. When a few errors did occur in CF, PUP6 itself had to be altered.

8. Conclusions

8.1. About PUP6

What have we learned from this experimental study? The overall feasibility of Beings was demonstrated, but the difficulties of communicating with the user made the system almost impossible to work with. The set of questions the user was expected to want to ask is the same as the set that one Being can ask another: the Being parts. When the "nice" user interrupts, his questions are translated trivially into a simple retrieval. Real users are seldom nice, the Beings generally misunderstood what users asked.

To modify PUP6 to synthesize new programs, it was necessary to add a few general-purpose programming and communication Beings, plus add several Beings specific to the new program's domain, plus generalize a few existing Beings' parts. The dialogue to produce the new program may be poorly suited to that domain, since most of the recognized phrases stem from a single (CF-producing) protocol.

To improve PUP6's performance, one could add some debugging specialist Beings, some dialogue specialists, some sophisticated user psychology experts (why is the user asking me that question, what needn't I tell him, how should I direct his attention), some Beings whose task is to aid the untrained user in inserting new domain-specific Beings, and perhaps a whole library of varied specialist Beings.

8.2. About Beings

The performance of the Beings representation itself in PUP6 is mixed. Two advantages were hoped for by using a uniform set of Being parts. Addition of new Beings to the pool was not easy (for untrained users) but communication among Beings was easy (fast, natural). Two advantages were hoped for by keeping the Beings highly structured. The

interactions (especially with the user) were brittle, but the complex tasks put to the pool were successfully completed.

The crippling problems are seen to be with user-system communication, not with the Beings ideas themselves. Sophisticated, bug-free programs were generated, after hours of fairly high level dialogue with an active user, after tens of thousands of messages passed among the Beings. Part of this success is attributed to distributing the responsibility for writing code and for recognizing relevance, to a hundred entities, rather than having a few central monitors worry about everything. The standardization of parts made filling in the Beings' contents fairly painless.

What are Beings good for? For which tasks won't the problems encountered in PUP6 recur? The idea of a fixed set of parts (which distinguishes them from ACTORS) is useful if the mass of knowledge is too huge for one individual to keep "on top" of. It then should be organized in a very uniform way (to simplify preparing it for storage), yet it must also be highly structured (to speed up retrieval).

For these reasons, the author is currently investigating, as a potential task domain, "research in elementary number theory". This has the added benefit of isolating the problems in representation research from the staggering complexities of natural language handling. Beings are big and slow, but valuable for organizing knowledge in ways meaningful to how it will be used. In the future Automated Mathematician system, Beings will be one -- but not the only -- internal mechanism for representing and manipulating knowledge.

References

- [1] Green, Waldinger, Baistow, Elschlager, Lenat, McCune, Shaw, and Steinberg, *Progress Report on Program-Understanding Systems*, Memo AIM-240, Stanford Artificial Intelligence Laboratory, August, 1974.
- [2] Hempel, Carl G., *Fundamentals of Concept Formation in Empirical Science*, Chicago: U. Chicago, 1952.
- [3] Hewitt, Carl, *A Universal Modular ACTOR Formalism for Artificial Intelligence*, Third International Joint Conference on Artificial Intelligence, 1973, pp. 235-245.
- [4] Minsky, Marvin, *Frames*, in (P. Winston, ed.) *Psychology of Computer Vision*, New York: McGraw Hill, 1975.
- [5] Newell, A., *Production Systems: Models of Control Structures*, in (W. Chase, ed.), *Visual Information Processing*, New York: Academic Press, 1973.
- [6] Teitelman, Warren, *INTERLISP Reference Manual*, XEROX PARC, Palo Alto, Ca., 1974.
- [7] Winston, Patrick, *Learning Structural Descriptions from Examples*, MIT Project MAC, TR-231, September, 1970.
- [8] Woods, W.A., and Makhoul, J., *Mechanical Inference Problems in Continuous Speech Understanding*, Third IJCAI, 1973, pp. 200-207.

Many hours of creative discussions were quite important. In particular, the author acknowledges C. Green, R. Waldinger, D. Shaw, and E. Sacerdoti. Computer time was generously provided by the Artificial Intelligence Center of SRI.

Appendix 1: A Typical Being

We consider INFO-OBTAINER, a Being which is independent of task domain. Below is listed, for each part, its abbreviated name (in bold), an English question that it might (try to) answer, and the stored program which should try to answer it (often a simple template or a constant). The percentages given indicate how many of the (ultimately 100) Beings in PUP6 actually used that part during the synthesis of one of the three target programs.

WHAT	What do you do? Summarize your basic idea.	82%
	(OBTAIN SOME INFORMATION WHICH CAN BE USED)	
WHY	Justification? Summarize your motivation.	77%
	(PUP HAS NO MORE INFO THAT IT CAN USE TO PROGRESS)	
HOW	Global strategy? Summarize your method.	72%
	(OBTAIN NEW FACTS ABOUT OLD INFO, OR BRAND NEW INFO)	
IDEN	Do you recognize: "Find out more about frob gyration"? (if I see either phrase: (INFO-OBTAINER any1) or (FIND OUT MORE ABOUT any1), then I return: (INFO-OBTAINER (TRANSLATE any1)))	54%
EXPLICIT-ARGS	What argument(s) do you take? (U)	63%
EVAL-ARGS	Which are quoted, not evaluated? NIL	4%
IMPLICIT-ARGS	What local variables are needed? NIL	11%
WHEN	When should you take control (justify your answer)? ((if T then add in -10 because (I AM EXPONENTIALLY-GROWING, GENERALLY UNDESIRABLE)) (if New-Info-List then add in (Plus 100 (Length New-Info-List))) because (WE SHOULD WORK ON UNASSIMILATED NEW INFORMATION IF THERE IS ANY)))	19%
REQUISITES	What must you ensure is true just before (pre) and after (post) you go? NIL	10%
DEMONS	What demons should you activate while you're in control? NIL	7%
META-CODE	What happens when you are in control? (DO (CHOOSE-FROM ((GET-NEW-INFORMATION U) (TRANSLATE U) (ANALYZE-IMPLICATIONS U) (EXTRACT-RELEVANT-SUBSET U))) BECAUSE (WE CAN ONLY TRY TO OBTAIN USABLE INFO IN ONE WAY AT A TIME))	70%
COMMENTS	Do you have any special hints for filling in undefined subparts of this Being? NIL	16%
STRUCTURE	Viewing this Being as a data structure, what can you do to it? NIL	4%
MAIN-EFFECTS	Can you cause this goal to occur: "Usable information exists"? ((to get (NEW INFORMATION any1) or to get (USABLE INFORMATION any1), do (INFO-OBTAINER any1)))	27%
AFFECTS	What other Beings might you call on directly? ((CHOOSE-FROM is called) (call on some Being who can satisfy the goal: (AWARE USER (ABOUT TO OBTAIN USABLE INFO)) (GET-NEW-INFORMATION possibly is called) (TRANSLATE possibly is called) (ANALYZE-IMPLICATIONS possibly is called) (EXTRACT-RELEVANT-SUBSET possibly is called))	14%
GENERALIZATIONS	What Beings are more general than you? (WRITE-PROGRAM SERVE-THE-USER)	27%
ALTERNATIVES	What Beings are similar to you, to try in case you fail? (USE-INFORMATION, OPTIMIZE, FIX-INCORRECT-PIECE, and FILL-IN-UNDEFINED-SECTION)	16%
ENCODABLE	Any special constraints on what order the parts should be eval'd in? NIL	9%
COMPLEXITY-VECTOR	How costly are you? (.5 .5 .9 .5 .1)	92%

A vector of utility measures. The first component says that INFO-OBTAINER is of average difficulty to call. Next, there exists a .5 chance that some descendant will call it again. Next: this activity almost always succeeds. The time/space used in allowing this Being to try is typical. Finally, there is no good reason for inhibiting it ever. In general, each component can be a *program*, not just a constant. These weights, like the contents of all the parts of all the Beings initially in the experimental PUP6 system, were decided upon and inserted by hand.